- COLLECTION –

Exception:

A collection is, in our words, a structure of data

- **List:** Similar to arrays but does not have to be fixed in size when it is first created and its size is flexible
  - ✓ ArrayList
  - ✓ Vector
  - ✓ LinkedList
- **Map:** An object that maps keys to values. A map cannot contain duplicate keys; each key can map to at most one value.
  - ✓ HashMap
  - ✓ properties

1. LIST

Array list: Contains duplicate elements.

- Maintains insertion order.
- Non synchronized (multiple threads can access methods of that particular class at any given time)
- Allows random access because the array works on an index basis.

Description	Command	Explaination
Constructing	`ArrayList<ObjectType> arrayListName = new ArrayList<String>();`	**Object type:** The type of the data that the list will hold, needs to be defined whilst constructing the list. This could either be a primitive type already defined (String, int…etc..) or it could be an object the user makes by making a class and defining its structure (Friend: name, tel, birthday; Car: brand, color, engine ; et..)

Adding values	COMBINED:
	`arrayListName.add("str0");`
	`arrayListName.add("str1");`
	`arrayListName.add("str2");`
	
	`arrayListName.add(1, "str111111111111");`
		Index:
		- When index is not specified it enters the values in the next index available (example: add("str0")).
		- When index is specified, it adds itself in that index and all the indexes after it gets pushed back by 1 (example: add(1, "str111111111111")

	SEPERATELY:
	`objName temp = new objName(str3);`
	`friendListName.add(temp);`	

Changing values	`arrayListName.set(1, "11111");`	Converts value at index 1 to 11111
Removing values	REMOVING SPECIFIC VALUES: `arrayListName.remove(1);`	Removes value at index 1

	DELETING ENTIRE ARRAYLIST
	`arrayList.clear();`
	`arrayList = null;`
		Deleting the object “arrayListName

Printing list	PRINTING ENTIRE LIST:
	`System.out.println(arrayListName);`
	
	PRINTING VALUES ONE BY ONE WITH A TAB IN BETWEEN:
	`for (String alist: arrayListName) {`	

	`System.out.print(alist + "\t");}`	

Vector: implements a dynamic array that means it can grow or shrink as required. Like an array, it contains components that can be accessed using an integer index.

- It is safer to use vector when there are multiple threads
- Works in the same way an arrayList works (`Vector<Object> vec = new Vector<Object>();`)

Linked list: Similar to an array list but additionally, it connects data that exists discontinuously.

2. MAP

Hashmap: A Map based collection class that is used for storing Key & value pairs, it is denoted as HashMap<Key, Value>

- It is not an ordered collection which means it **does not** return the keys and values in the same order in which they have been inserted into the HashMap.
- **Iterator (repeating a process):** It is used to retrieve elements one by one (in the Collection framework), when we want to print a s specific data in a map.

```
Iterator<Integer> iterator = hashamp.keySet().iterator();
while (iterator.hasNext()) {
Integer key = iterator.next();
System.out.println("data of "+ key + "is" + hashmap.get(key));
```

Description	Command	Explanation
Constructing	Ex.1: `HashMap<Integer, String> hashmap = new HashMap<Integer, String>();` Key: Integer Ex.2: `HashMap<String, Friend> hashmap2 = new HashMap<Integer, Friend>();` Key: String	- User chooses the key for the map (to differentiate and find the values). - And for each key, it adds a value of an object of its choice: (1) Primitive: String, int or (2) Constructed objects: friend, member, car…etc..

Adding values	`hashmap.put(11, "str11");`
	`hashmap.put(20, "str20");`
	`hashmap.put(8, "str8");`
	
	`hashmap2.put("A", new Friend("Anna", "2000/07/08"));`
	`hashmap2.put("B", new Friend("Ben", "1994/11/28"));`
		- The values within the map has no order. 11, 20 , 8 in hashmap and A, B in hashmap2 are indexes (key)

Removing values	`hashmap.remove(8);`	
	`hashmap.clear();` `linkedListName = null;`	
Printing list	`System.out.println(hashmap.get(20));` `System.out.println(hashmap);`	